Postproceso de rectificación de metapolígonos

Juan A. Puchol García, Juan M. Sáez Martínez, Rafael Molina Carmona
Grupo de Visión, Gráficos e Inteligencia Artificial (VGIA)
Depto. Ciencia de la Computación e Inteligencia Artificial
Escuela Politécnica Superior de Alicante. Universidad de Alicante
e-mail: puchol@dccia.ua.es

Resumen

Uno de los métodos de representación de objetos orgánicos en 3D son los llamados blobs. Un blob es una isosuperficie definida por una expresión de potencial de campo finito. Contando con dicha expresión hemos de obtener la representación del cuerpo de la forma más adecuada. Para la representación de cuerpos en 3D suelen usarse técnicas de poligonalización, que consisten en una obtención del modelo de fronteras de la forma. Existen numerosos algoritmos que resuelven dicho problema para el caso de los blobs, casi todos basados en el padre de todos ellos: *Marching Cubes*. Nuestro objetivo es el estudio de dicho algoritmo para la obtención de un postproceso que nos proporcione unos resultados más correctos, sin un excesivo incremento de su complejidad computacional. Finalmente generalizaremos el postproceso para que pueda aplicarse a una malla poligonal obtenida a partir de cualquier algoritmo de poligonalización de blobs.

Palabras clave: Blob, Metaballs, Metapolígono, Triangulación, Vóxel, Marching Cubes, Raytracing.

1 Introducción

J.F. Blinn creó un método algebraico general de modelado llamado *"Modelo Blobby"* en el cual podemos expresar un objeto 3D en términos de isosuperficie (superficie de densidad constante) en base a un conjunto de primitivas generadoras de campo [1]. La expresión general para calcular el potencial de campo de un blob en un punto *(x,y,z)* cualquiera del espacio es la que se observa en la ecuación 1.

$$V(x,y,z) = \left(\sum_{i=1}^{N} b_i e^{-a_i f_i(x,y,z)} \right) - T \qquad (1)$$

En la expresión del potencial de campo (1), a_i es la caída de campo de la primitiva i, b_i es la fuerza de campo de la primitiva i, T es el umbral del campo y N es el número de primitivas. Como podemos observar dicha ecuación es muy difícil de parametrizar puesto que no es lineal. Por tanto convertirla a una ecuación paramétrica que nos determinase un conjunto de puntos donde el valor del campo fuese nulo (condición de pertenencia a la superficie) sería muy complicado. Por tanto únicamente contamos con una expresión matemática que, dado un punto del espacio y un conjunto de primitivas nos dice si el punto queda dentro del campo (valor de $V>0$), fuera del campo (valor de $V<0$) o en su superficie (valor de $V=0$).

Como puede observarse, el exponente de dicha fórmula es una función $f(x,y,z)$. Dicha función viene definida por el modelo que estemos utilizando. Pueden ser: campos de supercuádricas [2], Metaballs [3] y Soft Objects [4].

Nuestro objetivo es la representación de la frontera del blob para su posterior utilización como herramienta 3D. Podríamos representarlo directamente con una técnica de trazado de rayos rastreando todo el volumen del paralelepípedo que encierra al blob (que puede calcularse con la posición geométrica de las primitivas que lo componen) y buscando los puntos que forman su frontera mediante la función de densidad. Dichos puntos se iluminarían calculando la normal en los mismos mediante la expresión del gradiente (pues es la normal del plano tangente a la superficie en el punto). Pese a ser una técnica sencilla y precisa presenta un alto coste computacional. Para solucionar este problema se suele poligonalizar la frontera del sólido obteniendo un metapolígono que encierre convenientemente el volumen del mismo. Un metapolígono se puede tratar de forma más rápida puesto que existen numerosas técnicas avanzadas de representación de los mismos que incluso se implementan de forma estándar en el propio hardware. Nuestro objetivo es desarrollar una técnica general para la obtención de un modelo de fronteras lo suficientemente preciso de un blob.

Para poligonalizar un blob existen numerosos algoritmos, la mayoría basados en *Marching Cubes*, el cual fue diseñado por *Lorensen y Cline* [5]. Dicho algoritmo consiste en cortar el volumen del paralelepípedo que encierra al blob en un número de cubos de igual tamaño. Posteriormente se hacen intersectar los cubos con la superficie del blob. Dependiendo de la posición de los vértices exteriores / interiores y de los cortes, se poligonaliza de una u otra forma. Existen 14 posibilidades de corte de un cubo con la superficie eliminando rotaciones, traslaciones y simetrías.

Marching Cubes y la mayoría de los algoritmos derivados del mismo tienen el problema común de que en todo momento se analizan cubos de igual tamaño sea cual sea la región que se esté estudiando. Esto produce que el algoritmo no obtenga un conjunto de triángulos que determine de manera óptima la forma. Existe una variante interesante del algoritmo basada en descomponer los cubos que cortan a la superficie en cinco tetraedros [8], que a su vez vuelven a cortar a la superficie, con lo que se obtiene un metapolígono más suavizado que con la técnica original. El problema de dicha técnica es que es equivalente a disminuir el tamaño de cubo, aumentando de forma global el conjunto de triángulos y por tanto generando concentraciones excesivas en zonas donde no es necesario. Lo que perseguimos es una técnica con una medida de precisión local que estudie cada región del blob con diferente nivel de detalle, determinando el modelo de fronteras más aproximado a dicha forma.

Existen diferentes formas de medir la adaptación de una malla poligonal a un sólido basadas en distintos criterios, como por ejemplo el que plantean H.L. de Gougny y M. S. Shephard [10] basado en diferencias de volúmenes con el sólido original. Dichas técnicas de similitud sirven precisamente para medir la adaptabilidad de los distintos algoritmos que estamos mencionando.

Existen otras propuestas de algoritmos no derivadas de *Marching Cubes* basadas en recorrer directamente la frontera del blob graduando la exploración de dicha frontera dependiendo de medidas de precisión locales [6]. Pese a que los resultados son excelentes en cuanto al estudio local de precisión, se producen otros problemas derivados que no han sido resueltos como el de la detección de huecos en el sólido o el de detección de formas separadas.

Igualmente existen propuestas de algoritmos genéricos de generación de mallas para distintos tipos de superficies [9] basadas en triangulaciones como la de Delaunay. El problema de dichas técnicas es que no se basan en información específica de la propia superficie de estudio por lo que en la mayoría de ocasiones incurren en costes bastante altos. Otras técnicas específicas de generación adaptativa de mallas para blobs como la "*Poligonalización de blobs con rectificación de precisión*" [11] pese a obtener buenos resultados presenta el inconveniente de la excesiva complejidad que supone su implementación, lo que reduce su aplicación práctica y su generalización. El algoritmo que proponemos se plantea como un postproceso de ajuste, con lo que será aplicable a cualquier poligonalización de blobs de forma sencilla. En los siguientes apartados explicaremos el procedimiento seguido en *Marching Cubes* y plantearemos el postproceso de ajuste sobre dicho algoritmo.

2 Algoritmo de Marching Cubes

Se trata de un algoritmo para la poligonalización de cualquier tipo de isosuperficies que consiste en lo siguiente: En primer lugar se determina el paralelepípedo que encierra al blob haciendo uso de la información de la posición y tamaño de las primitivas que lo componen. Seguidamente dividimos dicho paralelepípedo en cubos de tamaño constante. Dicho tamaño será definido por el usuario dependiendo de la precisión que desee para la figura resultante.

A continuación se analiza cada cubo por separado. Inicialmente hemos de formar un *cluster* de 8 bits que nos defina la colocación de cada cubo con respecto al blob. Dicho *cluster* se formará con la información de la pertenencia de los distintos vértices del cubo al blob de forma que los vértices interiores caracterizados por tener un potencial >=0 los marcaremos con un 1 y los exteriores que están caracterizados por tener un potencial <0 serán marcados con un 0.

Una vez tenemos la información sobre la colocación del cubo con respecto al blob procederemos a estudiar qué forma tendrá la porción de superficie asociada a dicho cubo. Para ello *Lorensen y Clyne* [5] estudiaron de cuántas maneras puede un cubo intersectar con una superficie. De las 256 posibles combinaciones (por la definición del cluster tenemos 2^8=256 combinaciones de los valores binarios), eliminaron los casos de colocación físicamente imposibles. Si además tenemos en cuenta simetrías y rotaciones entre los distintos casos el número disminuye a 14.

Por último, para cada una de las 14 combinaciones citadas se estudió la mejor forma de poligonalizarlas. La forma de dichas poligonalizaciones así como los posibles casos de intersección de un cubo con la superficie se pueden observar en la figura 1.

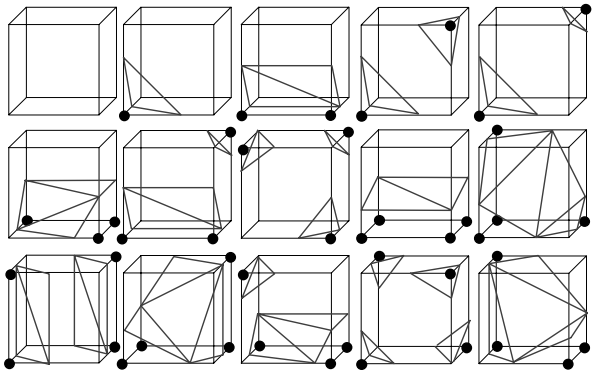


Figura 1: Casos de intersección de un cubo con un blob

2.1 Ventajas e inconvenientes del algoritmo

El algoritmo presenta grandes ventajas como puede ser su sencillez de implementación con respecto a otros. Por otro lado resulta computacionalmente sencillo (bajo coste) y la malla resultante es igualmente sencilla.

Los inconvenientes del algoritmo son muchos. El principal es que por propia naturaleza del mismo el rastreo del volumen se realiza al mismo nivel de detalle en todas las zonas, por lo que no se asegura la calidad final de la forma. Este problema es derivado de la escasa información que se toma de la superficie y la localidad de la misma. Por otro lado no se asegura un ángulo mínimo entre las normales de polígonos contiguos, con lo que la iluminación del metapolígono deja mucho que desear en la mayoría de los casos (efecto vóxel). Tampoco se asegura la coherencia entre el volumen real y el poligonalizado por lo que en algunos casos la conectividad del metapolígono no es la misma que la del sólido original. Finalmente no se especifica ninguna forma de determinar el tamaño de cubo óptimo para una determinada forma.

Estos problemas surgieron en la versión inicial del algoritmo, la cual ha sido modificada por muchos autores que persiguen su mejora desde distintas perspectivas. Por ejemplo en la versión inicial se encontraban huecos en la malla poligonal en distintos casos. Dicho error ha sido solucionado por varios autores añadiendo varios casos más de posibles intersecciones del cubo con el blob. Por otro lado el problema de la coherencia de los volúmenes, ha sido solucionado por varios autores, como es el caso del algoritmo "*Volume preserving MC*". Igualmente el problema del tamaño de cubo puede ser solucionado de forma sencilla calculándolo en base a la información geométrica de las primitivas, tomando dicho tamaño en función de la primitiva más pequeña que tenga la escena. De esta forma se asegura que ninguna primitiva se queda totalmente incluida por un cubo con lo que no obtendríamos intersección y perderíamos la porción de volumen.

El problema de la precisión y el de asegurar las normales entre polígonos contiguos es un problema más complicado. Podríamos introducir cualquier técnica de suavizado de aristas para solucionarlo pero el problema de estas técnicas es que son muy generales y normalmente al no contar con información del sólido original tienden a hacer el metapolígono aún más distinto a la forma real. Perseguimos una técnica específica que se ajuste a las características de los objetos orgánicos.

3 Postproceso de rectificación de metapolígonos

En este apartado realizaremos una descripción detallada del algoritmo que proponemos. Tenemos como objetivo particular la obtención de un postproceso de *Marching Cubes* lo suficientemente rápido como para no incrementar considerablemente el coste de éste y que resuelva algunos de los problemas del mismo. La idea es obtener una malla poligonal más precisa a partir de la malla poligonal que nos proporciona *Marching Cubes*. Dicho postproceso tendrá como objetivo asegurar la normal entre triángulos contiguos, aplicando criterios basados en información local. Por tanto, también resolveremos problemas relacionados con la pérdida de exactitud en zonas que requieran alto nivel de detalle. Nos centraremos por tanto en la rectificación de la malla poligonal de forma que en cada volumen parcial aumentaremos la concentración de polígonos en las zonas en las que necesitemos alto grado de detalle, respetando la malla original en aquellas zonas donde la calidad de la misma sea suficiente. Para introducir el algoritmo veremos inicialmente como puede ayudarnos la expresión del gradiente del blob.

3.1 El gradiente y su aplicación como medida de precisión

Como veremos en apartados posteriores el vector gradiente de la fórmula de densidad nos proporcionará gran cantidad de información aprovechable. El gradiente de una fórmula de densidad en un punto se puede definir como el vector normal del plano tangente a la superficie en dicho punto. En todos los procesos de ocultación e iluminación se utiliza la normal de los polígonos. Dicha normal se puede obtener del polígono directamente si todos los polígonos de la figura vienen expresados en un mismo orden dextrógiro o levógiro. Teniendo la normal en cada vértice mediante el gradiente, podemos hacer la media aritmética de las normales en los vértices para obtener la normal del triángulo directamente. Dicho vector nos servirá igualmente como medida de precisión de un triángulo pues nos marca lo bien o mal colocado que está éste. La expresión del gradiente puede observarse en la siguiente figura.

$$V'(x,y,z)=\left(\frac{\delta V(x,y,z)}{\delta x},\frac{\delta V(x,y,z)}{\delta y},\frac{\delta V(x,y,z)}{\delta z}\right) \tag{2}$$

Para cada triángulo perteneciente a la malla poligonal que nos calcula *Marching Cubes* podemos obtener el vector gradiente en todos sus vértices (v_1, v_2, v_3). Seguidamente obtenemos el ángulo que forman dos a dos los tres vectores

gradiente. Con ello obtenemos tres ángulos α_1(ángulo entre v_1 y v_2), α_2 (ángulo entre v_2 y v_3) y α_3 (ángulo entre v_3 y v_1). Definimos también un ángulo umbral ***U*** el cual simbolizará el valor máximo que pueden tomar dichos ángulos. Por tanto, la tripleta [α_1, α_2, α_3], nos marcará una medida de precisión de cada triángulo. El umbral nos servirá para marcar el grado de detalle que queremos para todos los triángulos que forman la malla poligonal.

Como medida adicional podemos igualmente marcar la longitud máxima **L** que puede tomar una arista perteneciente a un triángulo. De esta forma aseguramos que no habrá polígonos con aristas excesivamente alargadas. Esto nos servirá para que la malla resultante sea más homogénea y no se generen triángulos degenerados. Para ello tendremos igualmente un vector de longitudes [l_1, l_2, l_3] obtenido mediante el cálculo de las distancias euclídeas entre los pares (v_1, v_2), (v_2, v_3) y (v_3, v_1) respectivamente.

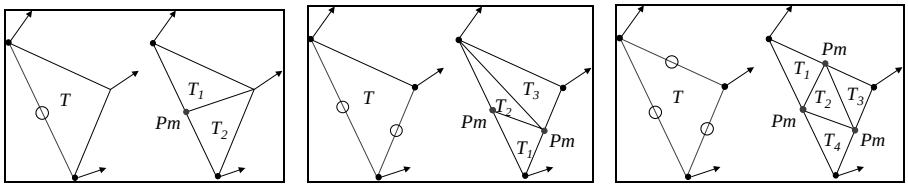
3.2 División poligonal de la malla

Una vez contamos con la medida de precisión de un triángulo podemos realizar un algoritmo que adapte la malla que nos calcula *Marching Cubes* hasta que todos los triángulos de dicha malla cumplan que el ángulo máximo entre cualesquiera dos vectores gradiente medidos en los vértices del mismo sea menor que ***U*** así como que las longitudes de las distintas aristas que lo componen no excedan **L**. El procedimiento será el siguiente:

```
Función DivisionPoligonal(MallaTriangular M, Umbral U, Dmáxima L):MallaTriangular
Var   MallaTriangular::M_final;
      Triángulo::T;
      Angulo::α1, α2, α3;
      Distancia:: l1 l2 l3; Finvar
  Para cada triángulo T perteneciente M hacer
      Obtener los tres vectores gradiente de T en sus vértices
      Calcular los tres ángulos posibles [α1, α2, α3] entre los vectores gradiente.
      Calcular las longitudes de las aristas del triángulo [l1, l2, l3]
      Si α1>U o α2>U o α3>U o l1>L o l2>L o l3>L entonces
          Eliminar T de M
          Añadir particiones de T a M según [α1, α2, α3] y [l1, l2, l3] y adaptarlas a la
          superficie
      Sino
          Eliminar T de M
          Añadir T a M_final
      Fsi
  Fpara
  Devolver M_final
```

Como puede observarse esta es una función recursiva puesto que sobre la misma malla insertamos las particiones de los triángulos, los cuales vuelven a ser procesados hasta que todos son aceptados por el criterio. Es conveniente determinar una altura suficiente del árbol de recursión, esto es, permitir un número de particiones máximo en un triángulo para no caer en un nivel de detalle excesivo.

El problema ahora consiste en dividir de forma eficiente los triángulos cuando no superan el criterio. Las particiones se realizarán de forma que se respeten aquellas aristas que superen el criterio dejando el resto de aristas del triángulo sin modificaciones. De esta forma, los cambios en aristas compartidas por polígonos vecinos se harán de la misma forma en ambos. Por otro lado, los nuevos vértices generados se toman en los puntos medios de las aristas implicadas. Los tres casos de partición posibles se pueden observar en la figura 2.



(a) (b) (c)

Figura 2: Casos de partición de triángulo, a) en dos, b) en tres y c) en cuatro triángulos.

En la figura anterior puede observarse que las posibles subdivisiones de un triángulo se reducen a tres eliminando simetrías. En el primer caso el triángulo únicamente se divide en dos pues solo falla una de las aristas. En el segundo caso fallan dos aristas con lo que se generan tres triángulos. El tercer caso es el más desfavorable puesto que fallan todas las aristas y el triángulo ha de dividirse en cuatro partes. Obsérvese que la subdivisión siempre se realiza tomando los puntos medios de las aristas que fallan como nuevos vértices de los triángulos que se generan, combinados con los vértices que habían en el triángulo original.

Utilizando estos criterios de partición y adaptando los nuevos vértices a la superficie como veremos a continuación, se consigue que los ángulos entre vectores normales de los nuevos triángulos decrezcan hasta cumplir el criterio.

Una vez dividido el triángulo el problema es adaptarlo convenientemente a la superficie del blob. Los vértices originales del triángulo dividido pertenecen a la frontera del blob pero los nuevos vértices obtenidos no tienen por qué cumplir esa

propiedad. Por ello tenemos que hacer coincidir los nuevos vértices con la frontera. En la figura 3 podemos observar un ejemplo de dicho ajuste.

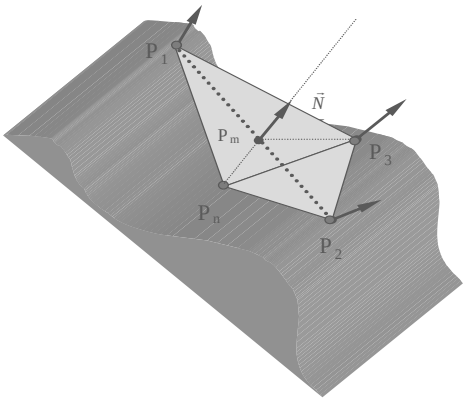


Figura 3: Ajuste del nuevo vértice producido por la división de un triángulo.

En esta figura el triángulo original está formado por los vértices P_1, P_2 y P_3 y los vectores gradiente en los puntos P_1 y P_2 forman un ángulo mayor al umbral (hay un valle bajo éstas) mientras que los ángulos entre los vectores gradiente de P_1 y P_3 así como P_2 y P_3 son correctos. Dividimos el triángulo en dos (primer caso): [P_1, P_m, P_3] y [P_3, P_2, P_m] siendo P_m el punto medio entre P_1 y P_2. Para ajustar el nuevo vértice (P_m) calculamos la normal en él (N) como la media de los vectores gradiente de P_1 y P_2. Como el valor del campo en P_m es negativo (exterior) nos movemos en dirección opuesta a la normal en la línea definida por el punto P_m y dicha normal hasta encontrar la nueva posición de P_m sobre la superficie del blob.

La razón por la que se toma la media de los vectores normales de la arista en vez de el propio gradiente de V en el punto medio de ésta es que es posible que intentemos calcular el gradiente dentro de una región de campo constante (puesto que el punto medio no tiene por qué estar contenido en la superficie) por lo que el gradiente se anularía y no sabríamos hacia dónde buscar el nuevo punto. Tomando la media de las normales en los vértices de la arista nos aseguramos de encontrar una dirección de búsqueda puesto que los tres vértices del triángulo original sí pertenecían a la superficie del blob y por tanto su gradiente no podía ser nulo. Una vez encontrado el punto en la superficie sí podemos asignar como normal del mismo el propio gradiente del campo en dicho punto.

4 Costes y ejemplos

En el siguiente apartado realizaremos una comparativa entre el algoritmo *Marching Cubes* y la rectificación propuesta sobre dicho algoritmo con respecto al coste. Inicialmente calcularemos el coste de forma teórica para después contrastarlo con las pruebas prácticas que se han realizado. Concluiremos con ejemplos sobre distintas escenas obtenidas con ambos algoritmos.

4.1 Coste teórico del método

Inicialmente calcularemos el coste de *Marching Cubes*:

Siendo V el volumen del paralelepípedo inicial que encierra al conjunto de primitivas y T el volumen constante de los cubos en los que se divide el mismo, se producen V/T iteraciones del algoritmo. Puesto que cada iteración se produce en un tiempo constante (calcular los cortes del cubo con el blob y producir la triangulación de los mismos) y T es constante, el procedimiento M.C. es lineal con respecto al volumen.

En cuanto al postproceso, se aplica a cada uno de los triángulos que genera *Marching Cubes*. Para cada cubo, el número de triángulos que pueden generarse mediante el algoritmo M.C. está entre 0 y 4 (figura 1). Por tanto, en el mejor de los casos en que ningún cubo genere triángulo alguno el postproceso no se aplicará y el coste seguirá siendo lineal con respecto al volumen. En el peor de los casos se producirán 4 triángulos por cubo, lo que suponen 4*V/T triángulos a los que se les aplicará el postproceso.

El postproceso realiza un número de iteraciones adicionales por triángulo que dependen de la propia colocación del mismo. Puesto que a partir de cierta altura en el árbol de división de los triángulos la segmentación resulta excesiva, introducimos una altura máxima del mismo a la que llamaremos H. Por tanto, en el peor de los casos se producirán 4^H iteraciones para cada triángulo puesto que como máximo se producen 4 particiones por triángulo (figura 2) y en el mejor de los casos ninguna. En el caso en el que no se produzca ninguna división para ningún triángulo, el número de iteraciones del algoritmo con la rectificación será igual que en el algoritmo M.C. En el caso de que se produzcan todas las particiones posibles de todos los triángulos, el número de iteraciones será $4*(V/T)*4^H$. Dado que H y T son constantes, el coste sigue siendo lineal con respecto al volúmen.

Finalmente podemos afirmar que si tanto para el mejor como para el peor de los casos el coste permanece siendo lineal, el algoritmo con el postproceso es lineal con respecto al volumen del paralelepípedo que encierra al blob.

4.2 Coste empírico del método

En la figura 4 se ha representado el tiempo de cálculo de ambos algoritmos para el mismo conjunto de muestras frente al volumen del paralelepípedo que encierra al blob. Sobre la misma gráfica hemos representado la recta de regresión lineal de cada serie para observar como afecta el postproceso en el tiempo de procesamiento de forma general.

Cada muestra se ha formado con un número de primitivas aleatorio entre 1 y 10. La posición de cada primitiva ha sido tomada también de forma aleatoria dentro de un volumen de tamaño 5x5x5. Para el tamaño de cada primitiva se ha tomado igualmente un valor aleatorio entre 0 y 4. Dichas medidas están referidas a un espacio tridimensional estándar de Open-GL. El umbral de ángulo tomado para la rectificación ha sido 25º. En todos los casos se han tomado primitivas esféricas.

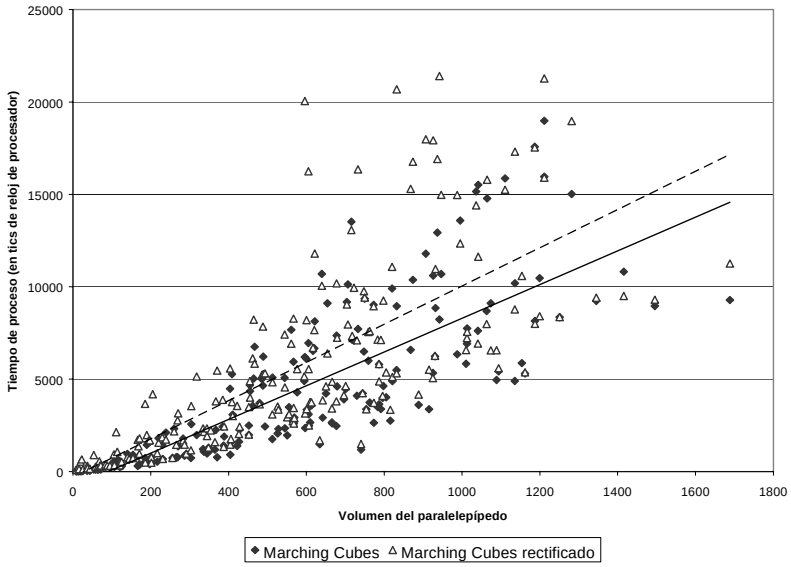


Figura 4: Representación del coste empírico de ambos algoritmos con sus rectas de regresión lineales.

Como vemos en la figura 4, las rectas de regresión de ambos algoritmos resultan similares, a excepción del incremento de pendiente del algoritmo rectificado con respecto al original. La magnitud del incremento de pendiente es inversamente proporcional al umbral que marquemos como ángulo máximo entre normales en la rectificación de la malla.

4.3 Ejemplos

En las figura 6 incluimos dos ejemplos representativos de ambos algoritmos. Nótese que se respeta la forma de la malla poligonal obtenida con *Marching Cubes* excepto en zonas donde tenemos que aumentar el número de polígonos para incrementar la calidad de la figura.

Igualmente puede observarse como se elimina el efecto vóxel al aplicar el postproceso, y como consecuencia se obtiene una forma mucho más definida. Esto es debido a que el postproceso aumenta la densidad de polígonos en las zonas donde los cambios de gradiente se acentúan subdividiendo los polígonos que se encuentran en las mismas.

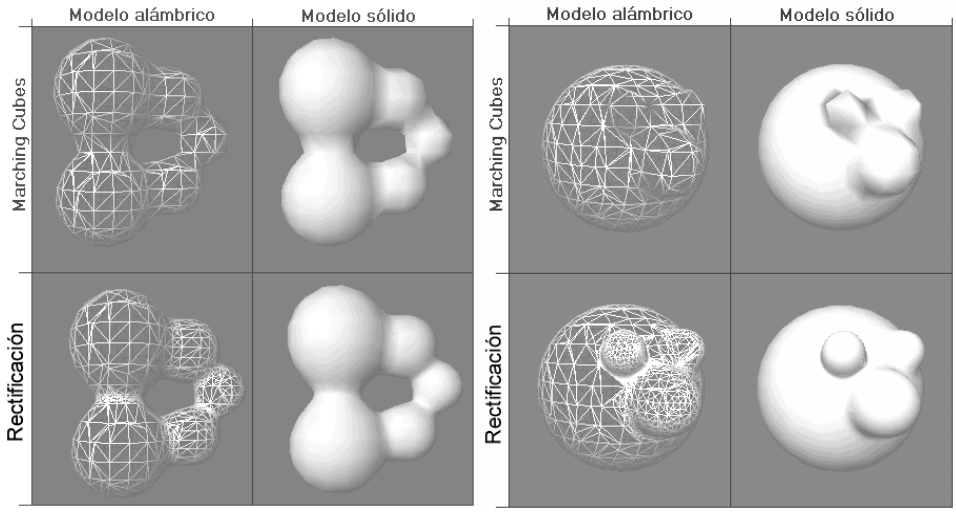


Figura 6: Ejemplos de funcionamiento de ambos algoritmos.

5. Generalización del método y conclusiones

El método propuesto es totalmente aplicable a cualquier algoritmo de poligonalización de blobs, puesto que se trata de un postproceso totalmente independiente del algoritmo original. Lo único que necesita el algoritmo es la malla poligonal resultante y la información de la superficie original. Por esta razón el algoritmo no sería aplicable directamente a un objeto poligonal cualquiera sin la información de la superficie que lo generó, pues sería necesaria la información del campo y del gradiente del mismo.

Igualmente, el algoritmo se puede generalizar como postproceso de algoritmos de poligonalización de otros objetos orgánicos además de los blobs, que sigan la misma filosofía de construcción, esto es: que estén basados en superficies de campo finito y que dichas superficies sean derivables.

La razón por la que se han elegido los blobs para el desarrollo de la técnica es porque son altamente representativos de los objetos orgánicos. Por otro lado, la razón por la cual se ha elegido Marching Cubes es que se trata de uno de los algoritmos de menor coste computacional de generación de mallas para isosuperficies. Esto nos ha permitido establecer comparativas sencillas entre el algoritmo simple y el algoritmo rectificado, sin entrar en otros problemas de coste que plantean otros algoritmos similares y que entorpecerían la labor de la estimación del coste del postproceso.

Referencias bibliográficas

1. Blinn, J.F.: “A Generalization of Algebraic Surface Drawing”, ACM Trans. On Graphics Vol 1, No. 3, pp.235-256, 1982.

2. Barr, A.H.; “Supercuadrics and Angle-Preserving Transformations”, IEEE Computer Graphics and Applications. Vol. 1, No 1, pp. 11-23, 1981.

3. Nishimura, H., Hirai, M., Kawai, T., Kawata, T., Shirakawa, I. and Omura, K.;”Object Modeling by distribution Function and a Method of Image Generation”, Trans.IEICE Japan, Vol J68-D, No. 4, pp. 718-725.

4. Wyvill, G., McPheeters, C. and Wyvill, B.; “Data structure for Soft Objects”, The Visual Computer., Vol. 2, pp. 227-234, 1986.

5. Lorensen, W.E. and Cline, H.E. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm”, ACM Computer Graphics, Vol 21, No. 24, pp. 163-169, Julio 1987.

6. Puchol García, J.A.; Molina Carmona, R.; García Quintana, C. "Conversión Paramétrica de Blobs a Modelo de Fronteras en Tiempo Real”. IX Congreso Internacional de Ingeniería Gráfica. Bilbao - San Sebastián, Junio 1997.

7. Requicha, A.A.G, and Rossignac, J.R.; “Solid Modeling and Beyond”, IEEE Computer Graphics and Applications, pp 31-44 1992.

8. B.A. Payne and A.W. Toga, “Surface mapping brain function on 3D models”, IEEE Computer Graphics and Applications, pp 33-41 1990.

9. H.Borouchaki, F. Hecht, E. Saltel and P.L. George “Reasonably efficent Delaunay based mesh generator in 3 dimensions” INRIA Report 1995.

10. H.L. de Cougny and M.S. Shephard “Surface meshing using vertex insertion”. Scientific Computation Research Center 1996.

11. Sáez Martínez J.M.; Puchol García J.A.; Molina Carmona, R. “Poligonalización de Blobs con Rectificación de Precisión” Departamento de Ciencias de la Computación e Inteligencia Artificial. Universidad de Alicante. Report DCCIA-98-04. Octubre de 1998.